http://www.applmathmech.cn

面向流体力学仿真的大型稀疏矩阵混合精度 GMRES 加速算法*

郑森炜[1]，　寇家庆[1,2,3]，　张伟伟[1,2,3]

(1. 西北工业大学 航空学院, 西安 710072;
2. 西北工业大学 流体力学智能化国际联合研究所, 西安 710072;
3. 飞行器基础布局全国重点实验室, 西安 710072)

(我刊青年编委寇家庆、编委张伟伟来稿)

摘要： 由于计算能耗低、效率高,以单精度/半精度计算单元为主的 GPU/TPU/NPU 等算力已成为人工智能计算的主要模式,但无法直接应用于浮点精度需求高的微分方程求解,不能直接替代双精度算力.通过结合单/双精度各自的优势,提出了兼顾效率和精度的大型稀疏线性方程组的混合精度求解格式.发展了面向稀疏大型矩阵的 GMRES 细化迭代算法(sparse GMRES-IR).首先分析了流体力学仿真问题中的矩阵数据分布特点,通过双精度做预处理,单精度细化迭代,使单精度计算应用于算法主要耗时部分,发挥了计算效率优势.通过求解开源数据集提供的 33 个线性方程组验证了所提出方法的精度和效率.结果表明,在单核 CPU 上,相同精度要求下,提出的单双混合精度算法可以实现最高 2.5 倍的加速效果,且在大规模矩阵下效果更突出.

关　键　词： 混合精度；　计算流体力学；　线性方程组；　GMRES
中图分类号： O35　　**文献标志码：** A　　DOI: 10.21656/1000-0887.450167

A Mixed-Precision GMRES Acceleration Algorithm for Large Sparse Matrices in Fluid Dynamics Simulation

ZHENG Senwei[1], KOU Jiaqing[1,2,3], ZHANG Weiwei[1,2,3]

(1. *School of Aeronautics, Northwestern Polytechnical University, Xi'an* 710072, *P.R.China*;
2. *International Joint Institute of Artificial Intelligence on Fluid Mechanics, Northwestern Polytechnical University, Xi'an* 710072, *P.R.China*;
3. *National Key Laboratory of Aircraft Configuration Design, Xi'an* 710072, *P.R.China*)

(Contributed by KOU Jiaqing, M.AMM Youth Editorial Board & ZHANG Weiwei, M.AMM Editorial Board)

Abstract: Due to low computational power consumption and high efficiency, GPUs/TPUs/NPUs with single/half-precision computing units make the main computing mode for artificial intelligence, but they can't be di-

* **收稿日期：** 2024-06-05; **修订日期：** 2024-07-10
作者简介： 郑森炜(2001—),男,硕士生(E-mail: senweiz@mail.nwpu.edu.cn);
寇家庆(1993—),男,教授,博士生导师(E-mail: jqkou@nwpu.edu.cn);
张伟伟(1979—),男,教授,博士生导师(通讯作者. E-mail: aeroelastic@nwpu.edu.cn).
引用格式： 郑森炜, 寇家庆, 张伟伟. 面向流体力学仿真的大型稀疏矩阵混合精度 GMRES 加速算法[J]. 应用数学和力学, 2025, **46**(1): 40-54.

rectly applied to solve differential equations requiring high floating-point accuracy, nor can they directly replace double-precision units. With the combined advantages of single and double precisions, a mixed-precision solution scheme balancing efficiency and accuracy, was proposed for large sparse linear equations. The sparse GMRES-IR algorithm for large sparse matrices was developed. Firstly, the characteristics of matrix data distributions in fluid dynamics simulation problems were analyzed. With double precision for pre-processing and single precision for detailed iteration, the single precision calculation was applied to the main time-consuming part of the algorithm, to enhance computational efficiency. Solutions of 33 linear equation systems from open-source datasets validate the accuracy and efficiency of the proposed method. The results show that, on a single-core CPU, under the same accuracy requirements, the proposed mixed-precision algorithm can achieve an acceleration effect of up to 2.5 times, and the effect is more prominent for large-scale matrices.

Key words: mixed-precision; computational fluid dynamics; linear equations; GMRES

0　引　　言

高性能计算硬件常用于解决科学、工程等计算领域的大规模问题.其中,计算流体力学(computational fluid dynamics,CFD)是高性能计算硬件的重要应用方向之一.随着计算复杂度与日俱增,CFD 对计算效率的要求也不断提升[1].图像处理器(GPU)等高性能硬件为求解复杂微分方程问题加速提供了新的可能.

近年来,国内外越来越多的硬件供应商推出了多精度浮点数混合计算单元架构的高性能计算硬件[2-3].如 Google 的张量处理单元(tensor processing unit,TPU),OpenAI 的神经处理单元(neural processing unit,NPU),都使用了低精度计算进行神经网络训练.这里的精度指的是计算机内部存储浮点数的精度,有 2 个基本类型:float(FP32)和 double(FP64),称为单精度浮点数和双精度浮点数.目前,许多硬件在设计中加入了半精度(FP16)浮点数计算单元,其内存大小仅为单精度浮点数的一半.一方面,低精度的计算单元可以在相同功耗下提供比高精度计算单元更强的算力[4].在 2017 年发布的 Tesla V100 GPU 上,半精度的矩阵处理速度是双精度的 4 倍[5].在 2020 年发布的 NVIDIA A100 GPU 上,单精度张量核心计算单元(TF32)的处理能力是双精度浮点计算单元的 8 倍左右;半精度计算单元(FP16 和 BF16)的处理能力则是双精度的 15 倍以上.这说明先进的高性能计算硬件的低精度浮点计算算力正在不断提高.另一方面,这些计算单元可以在不影响深度学习处理效果的前提下,大幅度降低功耗需求.与同等规模的系统相比,TPUv4 相比基于 NVIDIA A100 的系统,计算效率可提高 70%,同时功耗降低 50%[6].这展现了低精度浮点计算单元在高性能硬件改善计算效率和降低能源消耗上的潜能.

低精度浮点单元对大型网络训练性能的提升已经有显著成效[7-9].但是在实际工程问题中通常采用高精度计算,以满足实际问题的数值精度要求.基于低精度或混合浮点数计算单元的硬件无法在高精度计算中发挥出优势.因此,综合高精度浮点数计算和低精度浮点数计算的混合精度算法逐渐得到了重视[10].混合精度算法通过在算法的主要耗时部分采用低精度计算,在数据存储过程中采用高精度计算,结合了低精度算法的计算效率优势和高精度算法的准确性.在不影响结果精度的前提下,提高算法的计算效率.

目前,关于混合精度算法的研究问题,主要集中于求解线性方程组.混合精度算法进行线性方程组求解可以分为两个基本过程:预处理和细化迭代.预处理过程主要通过对原始的待求解矩阵进行矩阵分解,生成相应的预处理矩阵,以改善原始方程组的求解难度,提高求解效率.细化迭代部分首先通过一次求解后获得的解向量构造残差方程组,得到解向量的校正项,并对解向量进行修正.

根据细化迭代部分的求解方法不同,混合精度迭代算法可以分为直接求解法和迭代求解法.直接求解法通过对待求解矩阵进行三角分解或正交变换,可以在有限步数内获得方程的精确解;迭代求解法则通过给定初值,通过构造无穷序列来无限逼近精确解.Buttari 等[9]基于直接求解法构造混合精度算法.使用低精度的多前端稀疏求解器(MUMPS)[11]和 SuperLU 分解[12]做预处理.Hogg 和 Scott[13]设计了稀疏对称矩阵的混合精度求解器,采用低精度的 LDL 分解预处理,在单核 CPU 上,单双精度计算在超大规模的矩阵上起到了 2 倍的加速效果.

本文的求解算法以迭代求解法为基础.Carson 等[14] 提出了 GMRES-IR(generalized minimal residual with incomplete Richardson)算法,使用 GMRES 算法实现细化迭代过程,通过理论和数值实验证明所提出的算法收敛条件与待解矩阵的条件数相关.Higham 和 Pranesh[15] 通过数值实验,分析了半精度、单精度、双精度两两混合的 GMRES-IR 算法的收敛性质.Loe 等[16] 研究了 GMRES-IR 的参数调节方法.Amestoy 等[17] 研究了五精度的 GMRES-IR 的效果,并从条件数的角度研究了算法的收敛条件.Haidar 等[18] 在 GPU 平台上测试了 GMRES-IR 算法,使用了 NVIDIA 提供的 CUDA 库,发现半双混合精度的 GMRES-IR 算法在 GPU 平台实现了最大 4 倍的加速效果,但这仅仅是在大规模矩阵上的加速效果.

以上的算法都是基于稠密存储的矩阵数据.Zounon 和 Higham 等[19] 则研究了基于稀疏存储数据的 GMRES-IR 的加速效果,使用了低精度的 LU 分解作为预处理.结果表明,基于稀疏存储的 LU 分解算法在低精度计算中的加速效果不明显.原因在于对稀疏矩阵的分析和排序操作花费了大量的时间,而实际的浮点计算部分较少.上述研究的核心思想都来源于 Carson 等[14] 提出的 GMRES-IR 算法.本文所采取算法的基础也是如此.Gratton 等[20] 则提出了一种不精确的 Kroylov 子空间算法,并称为 GMRES-FD(GMRES float to double).Giraud 等[21] 和 Göbel 等[22] 分别以多重网格法和近似逆方法作为预处理,分析了其加速效果.

以往基于混合精度算法的线性方程求解器仍然面临一些待解决的问题.首先,由于稀疏矩阵的计算涉及大量的排序、分析和查找操作,实际的数值计算部分相对较少,混合精度算法的加速效果有限.其次,目前的研究对象主要集中在低条件数的任意方程组上,还没有针对工业问题中特定数值结构提出适用的求解方法.需要进一步的研究和探索.

在 CFD 求解过程中形成的 Jacobi 矩阵属于大型高度稀疏矩阵,使用传统的稠密矩阵算法会耗费大量内存和时间.本文在经典 GMRES-IR 算法的基础上,开发了一种更适用于 CFD 问题的线性方程组求解算法,即 sparse GMRES-IR 算法.该算法采用了高精度的不完全 LU(incomplete LU,ILU)分解作为预处理,低精度的 GMRES 算法进行细化迭代.在公开的稀疏矩阵数据集上进行数值实验.在单核 CPU 上,单双混合精度实现最高 2.5 倍的加速效果.

本文的算例测试平台基于单核 CPU,但设计算法理论可以拓展到多核系统及异构系统,并有望取得更大的加速效果.本文算法的核心方法是 ILU 分解算法和稀疏矩阵架构下的 GMRES 迭代方法.基于两种算法的并行实现已有相关研究.陈华等[23] 研究了 GPU 平台上的预处理稀疏矩阵 GMRES 算法并行效果,并与 CPU 算力结果进行对比,获得了 3~10 倍的加速效果.同时,由于混合精度算法的主要耗时部分通过低精度浮点数运算实现,随着高性能计算硬件的低精度浮点数处理能力的增强,理论上,本文所发展的算法将有更大的加速效果.

1 数 值 方 法

1.1 隐式时间推进方法

对于本文所关心的 CFD 问题,其求解的控制方程,可以描述为如下的半离散形式[24]:

$$\frac{\partial \bar{\boldsymbol{Q}}}{\partial t} + \boldsymbol{R}(\bar{\boldsymbol{Q}}) = \boldsymbol{0}, \tag{1}$$

其中空间项为离散格式,而时间项仍为连续导数形式.此处 $\bar{\boldsymbol{Q}} = \boldsymbol{J}^{-1}\boldsymbol{Q}$,$\boldsymbol{Q}$ 为守恒变量,$\boldsymbol{J}$ 是空间变换下的 Jacobi 矩阵.对于定常问题,时间导数项最终收敛到 0,可以忽略时间导数,离散方程可以简化为

$$\boldsymbol{R}(\bar{\boldsymbol{Q}}) = \boldsymbol{0}. \tag{2}$$

定义 $\Delta\bar{\boldsymbol{Q}}_i^n = \bar{\boldsymbol{Q}}_i^{n+1} - \bar{\boldsymbol{Q}}_i^n$,表示一次推进过程前后的差量.对于 NS 方程,式(2)中的离散方程为非线性方程,通常需要对左端项做线化处理.基于 Taylor 展开可以得到

$$\boldsymbol{R}(\bar{\boldsymbol{Q}}_i^{n+1}) = \boldsymbol{R}(\bar{\boldsymbol{Q}}_i^n) + \boldsymbol{A}_i^n \Delta\bar{\boldsymbol{Q}}_i^n, \tag{3}$$

其中 $\boldsymbol{A}_i^n = (\partial \boldsymbol{R}/\partial \boldsymbol{Q})_i^n$ 为通量 Jacobi 矩阵.

为了确保算法的稳定性,可以在式(3)左端引入伪时间项.伪时间项区别于物理时间,随着推进过程伪

时间项收敛到 0, 方程退化回式(3).此处可以不用考虑伪时间项的离散阶数, 采用运算成本较低的 Euler 法离散:

$$\frac{\Delta \bar{\boldsymbol{Q}}_i^n}{\Delta \tau} + \boldsymbol{R}(\bar{\boldsymbol{Q}}_i^{n+1}) = \boldsymbol{0}, \tag{4}$$

其中 $\Delta \bar{\boldsymbol{Q}}_i^n / \Delta \tau$ 便是引入的伪时间项,将式(4)代入式(3)中,整理可得

$$\left(\frac{\boldsymbol{J}}{\Delta \tau} + \left. \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}} \right|^m \right) \Delta \boldsymbol{Q}^m = - R(\boldsymbol{Q}^m) . \tag{5}$$

式(5)为 CFD 求解格式,该式中的待求解矩阵通常会有以下特点:

① 对角占优性.隐式推进方程由于引入了伪时间项,增大了主对角线上元素的大小.因此矩阵通常有对角占优性质,这改善了求解的稳定性.

② 高度稀疏性.通量 Jacobi 矩阵中绝大部分值都是 0.鉴于常用离散方法,如有限差分法和有限体积法,其离散化过程仅涉及相邻单元的通量传递,从而使得矩阵中绝大多数元素值为零.

上述的数据的两个特点决定了本文算法架构.由于矩阵是高度稀疏的,为了提高运算效率,降低计算机的内存需求,必须采用基于稀疏存储的算法;由于对角占优性,原始方程组的求解条件良好,即使采用低精度的迭代算法仍然可以收敛.这保证了混合精度算法在 CFD 问题上的适用性.

1.2　细化迭代方法

线性方程组求解问题可以表示为 $\boldsymbol{Ax} = \boldsymbol{b}$, 其中 $\boldsymbol{A}$ 为待求解矩阵, $\boldsymbol{b}$ 为右端项, $\boldsymbol{x}$ 为待求解向量.在考虑计算机舍入误差的情况下,其精确解可以表示为如下形式:

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} = \boldsymbol{x}^* + \boldsymbol{d}, \tag{6}$$

其中 $\boldsymbol{d}$ 为浮点数精度产生的舍入误差, $\boldsymbol{x}^*$ 为计算机实际求解出的解向量.定义残差 $\boldsymbol{r} = \boldsymbol{Ax}^* - \boldsymbol{b}$, 结合式(6)可得

$$\begin{cases} \boldsymbol{A}^{-1}\boldsymbol{r} = \boldsymbol{x}^* - \boldsymbol{x} = \boldsymbol{d}, \\ \boldsymbol{Ad} = \boldsymbol{r}. \end{cases} \tag{7}$$

即通过求解残差方程组 $\boldsymbol{Ad} = \boldsymbol{r}$, 可以得到待求解向量的修正量,且每当通过求解残差方程组对待求解向量更新后,都可以获得一个新的残差方程组.该方程组的待求解矩阵与原始方程相同,而右端项为更新后的方程残差.通过多次迭代求解残差方程,即可逐渐获得更高精度的解向量.该思想称为细化迭代思想.整个细化迭代过程如下:

① 给定初值 $\boldsymbol{x}$, 获得初始的残差 $\boldsymbol{r} = \boldsymbol{Ax} - \boldsymbol{b}$;

② 通过求解残差方程组 $\boldsymbol{Ad} = \boldsymbol{r}$, 获取修正量;

③ 更新修正后的解向量 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}$;

④ 重复步骤①—③,直到方程残差 $\| \boldsymbol{Ax} - \boldsymbol{b} \|$ 达到精度要求.

对于线性方程组求解问题,其所能达到的最高精度,取决于计算机可以有效存储的最高精度,称为机械精度.浮点数以二进制形式存储,标准形式表示为

$$V = 2^E \times (-1)^S \times M. \tag{8}$$

浮点数存储由三部分组成:符号位 S、有效位 M、指数位 E.机械精度由有效位和指数位的数值范围共同决定,但通常取决于有效位数的大小.因为指数位可表达的数值范围大小远超有效位.表 1 展示了 IEEE754 标准下的单精度和双精度的有效位数和数值范围.可以看到单精度浮点数的最小可表达的数值大小为10^{-45}, 而双精度浮点数的有效位数仅 16 位.对于归一化后的数据而言.单精度浮点的有效位精度仅仅到10^{-7},而双精度仅到10^{-16},远远小于指数位的数值范围.

在细化迭代方法中,求解残差方程组时,无论采用高精度算法还是低精度算法,最终都可以实现修正效果,原因在于所求解修正向量的量级通常远小于原始解向量的量级.此时,有效位的精度会进一步提高.例如若修正向量内的每一个数字的量级都不超过10^{-9},则对于单精度浮点数,其有效精度为10^{-16}.达到和归一化后的双精度浮点数据相同的有效精度.因此,通过低精度算法计算出的修正向量,仍然可以对原始解向量进

行修正.

表 1 IEEE754 双精度和单精度的有效位数和取值范围

Table 1 IEEE754 FP64/FP32 significant digit and value ranges

floating-point type	significant digit	maximum positive value	minimum positive normal
FP64	16	1.80E+308	4.94E−324
FP32	7	3.40E+38	1.40E−45

细化迭代方法的收敛条件主要与待解矩阵的病态程度有关,具体可以参考 Carson 和 Higham[14] 的工作.

1.3 稀疏矩阵 GMRES 细化迭代算法(sparse GMRES-IR)

GMRES 法是求解大规模稀疏矩阵方程的常用算法,具有收敛速度快,不用显式存储预处理矩阵,保证矩阵的稀疏性质等特点[25],常用于流体仿真及复杂微分方程的数值求解过程[26-27].其核心思想是通过构造 Krylov 子空间下的一组标准正交基,然后通过最小二乘法使残差最小,以获得解向量.所构造的 Krylov 子空间如下:

$$K_m = \mathrm{span}\{\boldsymbol{r}, \boldsymbol{A}\boldsymbol{r}, \boldsymbol{A}^2\boldsymbol{r}, \cdots, \boldsymbol{A}^{m-1}\boldsymbol{r}\}. \tag{9}$$

当方程的待求矩阵 $\boldsymbol{A}$ 的条件数 $\mathrm{cond}(\boldsymbol{A}) = \|\boldsymbol{A}\|_\infty \|\boldsymbol{A}^{-1}\|_\infty$ 过大时,此时称为矩阵病态.原始的 GMRES 算法会收敛较慢[25],需要通过预处理方法加速收敛.预处理方法是构造一个原始方程组的同解方程,通过预处理矩阵来改善原矩阵 $\boldsymbol{A}$ 的病态程度,以达到加速收敛的效果.本文采用的预处理方法为右预处理,其同解方程的表达式为

$$\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{M}\boldsymbol{x} = \boldsymbol{b}. \tag{10}$$

为了保证同解方程组的病态性更良好,构造了预处理矩阵 $\boldsymbol{M} \approx \boldsymbol{A}$.常用的预处理方法包括矩阵分裂法和矩阵分解,例如 Jacobi 预处理因子、ILU 分解、不完全 QR 分解等.本文采用 ILU 分解作为预处理方法.

经典的 GMRES-IR 算法[15] 主要解决稠密存储下的线性方程求解问题,算法 1 展示了其基本流程.经典 GMRES-IR 算法采用了低精度的 LU 分解作为预处理格式和高精度的 GMRES 进行细化迭代.低精度算法主要对 LU 分解过程实现加速,在稠密存储下,对矩阵进行 LU 分解的时间复杂度为 $O(n^3)$,其中 n 为矩阵的大小.当矩阵规模足够大时,GMRES 迭代的时间远低于 LU 分解所需要的时间.因此,采用低精度的预处理算法可以起到有效的加速效果.

CFD 中处理的矩阵方程组,其待求解矩阵有高度的稀疏性,因此通常会采用稀疏矩阵的存储形式来保存矩阵数据.经典的稀疏存储方式包括坐标列表格式(coordinate list,COO)、列压缩格式(compressed sparse column,CSC)、行压缩格式(compressed sparse row,CSR).COO 格式采用了三元组的形式来保存非零元素的位置和大小.CSR 格式则在 COO 格式的基础上,对非零元素的行列表进行压缩.本文采用的是 CSR 格式.

算法 1 经典 GMRES-IR 算法

input: matrix $\boldsymbol{A}_{n\times n}$; vector $\boldsymbol{b}$

output: solution vector of linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

1. LU factors of $\boldsymbol{A}$: $\boldsymbol{A} \approx \boldsymbol{M} = \boldsymbol{L}_\mathrm{f}\boldsymbol{U}_\mathrm{f}$ **lower precision**
2. solve $\boldsymbol{M}\boldsymbol{x}_0 = \boldsymbol{b}$, store $\boldsymbol{x}_0$ as working precision **lower precision**
3. store $\boldsymbol{L}_\mathrm{f}$, $\boldsymbol{U}_\mathrm{f}$ as working precision $\boldsymbol{L}_\mathrm{w}$, $\boldsymbol{U}_\mathrm{w}$
4. for $i = 0, \cdots, i_{\max} - 1$ do
5. $\quad \boldsymbol{r}_i = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_i$ **higher precision**
6. solve $\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{M}\boldsymbol{d}_{i+1} = \boldsymbol{r}_i$ by GMRES **higher precision**
7. if $\|\boldsymbol{r}_i\|_2 = \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_i\|_2 < \varepsilon_{\mathrm{tol}}$ then
8. $\quad \boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{d}_{i+1}$ **higher precision**
9. else
10. $\quad$ break
11. end if

12. end for.

如前文所述,混合精度算法的加速关键在于通过低精度浮点数计算算法中的主要耗时项以实现加速计算,而存储和其他部分采用高精度算法,保证最终结果达到精度要求.虽然经典 GMRES-IR 算法在应用于稠密存储的情况下表现出显著的性能结果.然而,由于以下原因,当采用稀疏格式存储的矩阵时,其加速性能无法保障.

① 完全的 LU 分解会破坏矩阵稀疏格式,导致在大型问题上出现内存不足的问题.即使原始矩阵是高度稀疏矩阵,LU 分解后的上三角矩阵和下三角矩阵却可能是稠密的.由于稀疏存储方式只需要以向量形式存储非零元素的位置信息和数值信息,不需要存储整个大型的二维矩阵,因而构造稠密的预处理矩阵需要付出巨大的内存代价.

② 基于稀疏存储的算法程序在低精度浮点数计算下的加速效果有限.在相同平台上,即不考虑通信时间的情况下,低精度计算基本仅在基本四则运算及由基本运算衍生的算法(例如矩阵乘法)上有加速效果.LU 分解需要对原始矩阵进行分析和变换,以保证原始矩阵的对角线上的元素皆不为零.在稀疏矩阵架构下,矩阵分析和变换算法中涉及大量的查找和重排序算法,这些算法无法在低精度上实现加速效果.因而,稀疏存储下 LU 分解在低精度浮点数计算单元上的加速效果有限.

对于上述原因,可以采用高精度稀疏架构下的不完全矩阵分解方法来构造预处理矩阵,例如 ILU 分解.ILU 通过预先假定矩阵中非零元素的位置和数量同原始矩阵相同,每次分解过程仅计算非零元素上的数值,由此保存了矩阵的稀疏架构,ILU 分解的基本流程如算法 2 所示.然而,ILU 分解是一种近似预处理方法,对 GMRES 收敛加速的效果有限,细化迭代的时间消耗增加.因而,基于稀疏存储的预处理方法并不适用于经典 GMRES-IR 算法.

算法 2　IJK 格式的 ILU 分解

input: sparse matrix $\boldsymbol{A}_{n\times n}$; nonzero element position P

output: ILU factorization L

1. for $i = 2,\cdots,n$ do
2. for $k = 1,\cdots,i-1$
3. if (i,k) in P
4. $a_{ik} = a_{ik}/a_{kk}$
5. for $j = k+1,\cdots,n$
6. if (i,k) in P
7. $a_{ij} = a_{ij} - a_{ik}a_{kj}$
8. end if
9. end for
10. end if
11. end for.

图 1 展示了本文所提出的基于稀疏矩阵存储架构的混合精度迭代算法,包括预处理部分和细化迭代部分.其中预处理部分采用了 ILU 分解作为预处理算法,构造高精度的预处理矩阵.并通过低精度算法进行细化迭代,将细化迭代的结果转化为高精度存储后,再对原始的解向量进行更新.这是为了保证最终的精度可以达到高精度浮点数要求.

总体而言,和经典算法相比,sparse GMRES-IR 算法具有以下特点和改进:

① 采用高精度的预处理方法和低精度细化迭代的混合精度算法.由上文所述,对于稀疏矩阵的预处理方法,如 LU 分解,可能无法在低精度上实现加速效果.为了不破坏矩阵的稀疏存储架构.在整个求解过程中,稀疏矩阵分解方法所需的计算时间相对较少,而细化迭代部分则成为主导因素.在这种情形下,采用低精度算法加速细化迭代部分是可行的.

② 基于稀疏存储的算法架构.本文的算法全程未破坏矩阵的稀疏存储结构,更有利于解决大型的稀疏

矩阵问题,并降低了对计算机内存的消耗,提高了计算效率.

③ 不完全的预处理方式.本文算法采用了 ILU 分解进行预处理.即预处理矩阵和原矩阵有相同的非零元素位置.这保留了矩阵的稀疏结构,但导致低精度算法难以起到加速效果.因而,本文算法的预处理过程采用了高精度计算.

④ 基于 CFD 问题的应用背景.本文算法针对 CFD 问题中产生的稀疏矩阵进行测试.数据具备两类基本特点,高度稀疏性和对角占优性(当时间步长足够小的时候).高度稀疏性意味着必须使用基于稀疏存储的算法;而对角占优性意味着求解条件良好,即使采用低精度的迭代算法也可以保证求解收敛,这保证了混合精度算法在 CFD 问题上的适用性.对于任意矩阵的 LU 分解或 ILU 分解而言,在进行矩阵分解时需要进行分析和预处理以保证算法的稳定性,这在稀疏存储架构下的矩阵分解运算中会带来额外的计算成本[19].由于 CFD 的时间推进矩阵具有对角占优性,在进行矩阵分解时,可以省略原本的矩阵分析部分,直接进行分解,降低了时间成本.然而,对于任意的稀疏矩阵,本文算法是否能够有效实现加速仍需进一步研究.

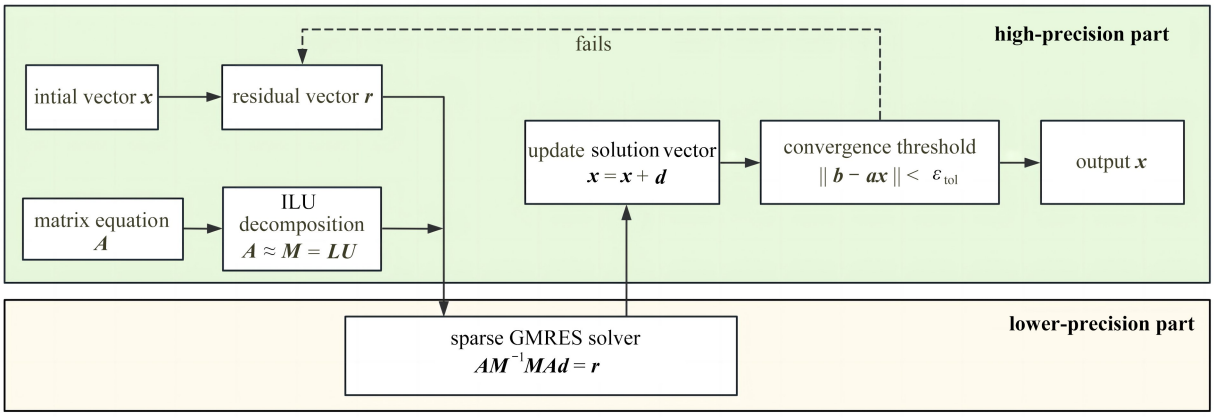


图 1 Sparse GMRES-IR 流程图

Fig. 1 The sparse GMRES-IR program flowchart

2 数值实验结果及分析

2.1 稀疏矩阵测试数据集

本文在单核 Intel i7-7700 CPU 上进行测试.测试程序使用 FORTRAN 90 编写.为了保证算例维度的多样性,数据采用佛罗里达大学的公开稀疏矩阵数据集(SuiteSparse matrix collection),该数据集包含了来自不同工业领域问题中的稀疏矩阵,本文选取了 33 组 CFD 相关背景的矩阵进行测试,右端项采用全 1 向量.矩阵的维度大小、非零元素数量、条件数以及稀疏程度的统计数据如表 2 所示.其中稀疏密度的定义为矩阵中非零元素的数量与总元素数量的比值,即非零元素数量与矩阵维度的平方之比.从统计结果可以看出,所选取的稀疏矩阵的密度均在 2%以下,且最小达到 0.004%.这意味着,若采用稠密矩阵的形式存储这些矩阵,所需要的计算机存储是稀疏存储的数十倍乃至数万倍.从数据范围可以看出,所选矩阵的数值范围均在单精度可表示的范围内,不存在溢出风险.

图 2 展示了稀疏矩阵的非零元素分布情况,可以观察到矩阵的非零元素分布呈现对角占优的特性,即对角线上的非零元素密度和大小远高于非对角线.这与前文描述的 CFD 问题中的稀疏矩阵特点相一致.条件数反映了矩阵在数值计算中的稳定性,对迭代法的收敛性和收敛速度具有重要影响.通常情况下,条件数越大,矩阵求解的难度越大,收敛速度越慢,本文选取的矩阵的条件数分布较广,确保算例数据的多样性.本文采用的条件数计算形式如下:

$$\mathrm{cond}(\boldsymbol{A}) = \| \boldsymbol{A} \|_{\infty} \| \boldsymbol{A}^{-1} \|_{\infty}. \tag{11}$$

本节设置双精度 GMRES 最大内迭代次数为 300,最大重启次数为 2,总迭代次数最大为 600.混合精度 GMRES 内迭代最大迭代次数为 100.对于细化迭代方法,低精度求解器的误差较大,不用要求每一步都完全精确求解,因而设置了比双精度迭代更少的最大迭代次数.外迭代的细化迭代次数最大为 10 次,这确保了混

合精度算法能最终达到残差收敛要求,收敛残差指标为10^{-11}.最终的结果精度用残差的均方根误差(RMSE,$\varepsilon_{\mathrm{RMSE}}$)表示,定义为

$$\begin{cases} \varepsilon_{\mathrm{RMSE}} = \dfrac{1}{\sqrt{N}} \| \boldsymbol{Ax} - \boldsymbol{b} \|_2, \\ \| \boldsymbol{r} \|_2 = \sqrt{\sum_{i=1}^{N} r_i^2}, \end{cases} \tag{12}$$

其中 $\| \boldsymbol{r} \|_2$ 表示对向量求 2 范数, r_i 是向量 $\boldsymbol{r}$ 中的相关元素.

表 2　测试算例中使用的矩阵参数信息

Table 2　The distribution of all parameters of the matrices used in the test case

	matrix dimension	non-zero elements count	condition number	sparsity ratio/%	absolute value range
maximum	109 460	2 374 949	5.63E12	1.979 938	3.57E10
minimum	1 080	14 133	885.246 1	0.004 111	−1.74E−10

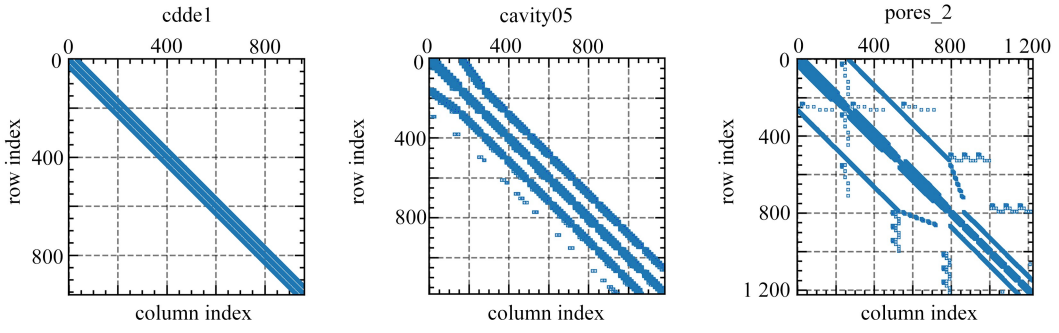


图 2　稀疏矩阵非零元素分布

Fig. 2　Distribution of non-zero elements in sparse matrices

注　为了解释图中的颜色,读者可以参考本文的电子网页版本,后同.

图 3 对比了混合精度算法与双精度算法的结果精度.结果表明,混合精度算法在多数情况下能够达到与双精度算法相当的精度水平,在其中 1 个算例(ex20)上,出现了双精度算法收敛而混合精度算法未收敛的情况.这说明混合精度算法基本可以满足精度需求,但在鲁棒性上不如双精度算法.

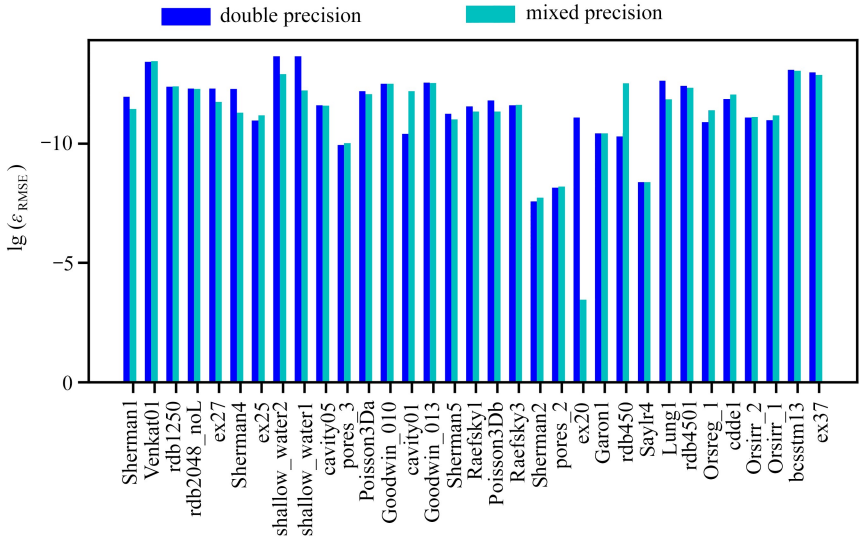


图 3　混合精度算法和双精度算法 RMSE 比较

Fig. 3　Comparison of RMSE between mixed-precision and double-precision algorithms

在一些算例上,混合精度的结果精度甚至略高于双精度算法,产生这一现象的原因与混合精度算法的算法架构有关,由于低精度的内迭代过程中计算的方程残差与高精度下计算的方程残差不一致,不适合作为收敛判据.因而,在本文的架构中,混合精度算法需要经过一次完整的内迭代过程后更新一次解向量,并根据更新后的解向量判断是否满足收敛要求,这可能导致内迭代的次数会高于达到目标精度的最低需求,使最终结果超过目标精度.

图 4 展示了混合精度算法的加速效果,其中蓝色柱条为双精度求解器的求解精度,绿色柱条为混合精度求解器的求解精度.在所测试的 33 个矩阵算例中,混合精度算法在 21 个矩阵上实现了加速效果,并达到了 10^{-11}次方量级,最大加速比达到 2.5 倍.这说明了混合精度算法在方程求解问题上有很大的加速潜能.我们观察到,在所测试的矩阵算例中,混合精度算法对部分矩阵的加速效果并不明显,甚至在个别情况下出现了计算速度下降的现象.在进一步研究中,我们发现加速效果不佳的算例往往是规模较小、计算时间较短的矩阵.这一现象将在下一小节中进行具体的讨论.

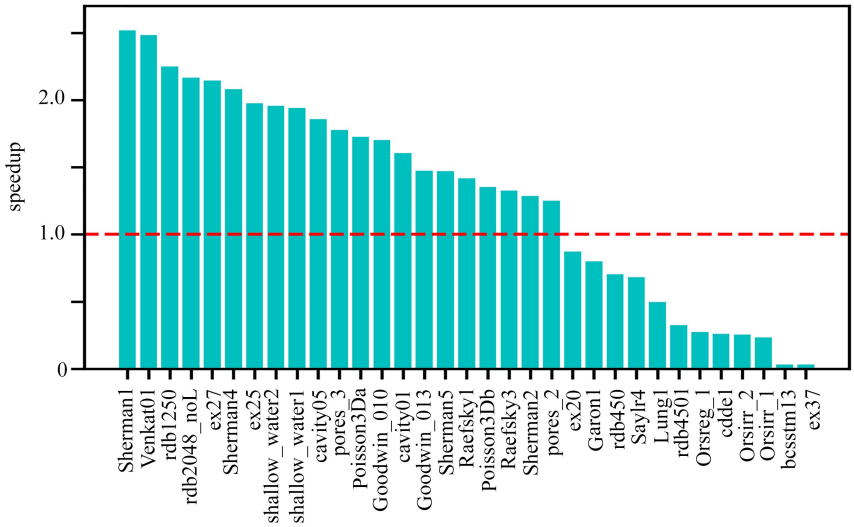


图 4 混合精度方法的加速效果

Fig. 4 Acceleration ratios of mixed-precision algorithms

2.2 计算加速性能分析

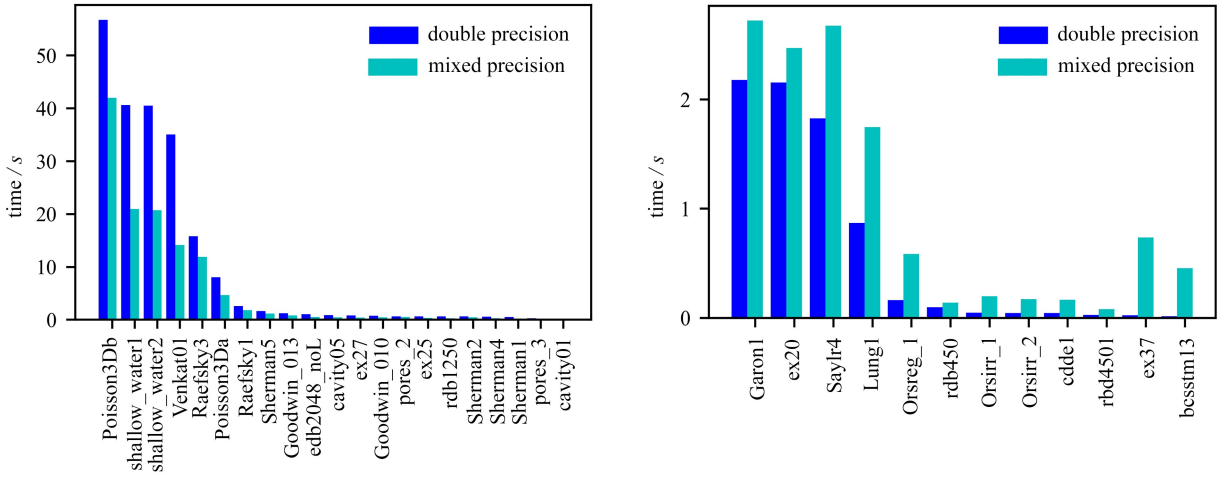
本小节将重点分析影响混合精度算法加速效果的主要因素,并给出可参考的判据来估计算法的加速效果.由于求解器的性能与迭代算法、矩阵条件数、预处理方式、收敛阈值有关,难以对混合精度算法性能进行一般化的说明.因此,本文采用数据挖掘方法给出了一个相对可供参考的加速指标,即在求解时间非特别短的情况下,当矩阵的条件数小于10^6 时,混合精度算法基本可以实现加速效果,且在大型矩阵上的加速效果更佳.

图 5 展示了不同算例中混合精度和双精度算法的计算时间.为了更好地区分不同加速效果上的矩阵区别,本文按照是否起到加速效果将算例分为两部分.图 5(a)展示了成功实现加速效果的算例,图 5(b)则展示了未实现加速效果的算例.并按照求解时间进行了排序.整体来看,在几个大规模的矩阵问题上,混合精度算法都起到了明显的加速效果.在大规模算例(如矩阵 Venkat01)上实现了超过了 2 倍的加速效果.而没有加速效果的算例的计算时间普遍较短.这说明了混合精度算法在大规模矩阵求解问题上有更好的性能.

图 6 展示了不同算例上双精度求解时间和加速比之间的关系.其中红色虚线是加速比为 1 的参考线.横坐标表示双精度求解器的求解时间,纵坐标为加速比.结果表明,在小规模矩阵问题上,混合精度算法普遍未能实现加速效果.本文推测这与数据的计算量和迭代次数较小,低精度浮点数未能发挥出优势性有关.在大规模矩阵问题上,混合精度算法可以普遍实现加速.这说明了混合精度算法可能仅适用于大规模矩阵的运算加速.在小规模问题上性能可能不如双精度算法,在中等规模的矩阵上还存在着鲁棒性的问题.

预处理和求解时间的占比对混合精度方法的加速比有重要的影响.对于稠密矩阵,完全的预处理方法

（如 LU 分解）可以有效地加快迭代方法的收敛速度，预处理过程的矩阵分解时间占据主导.因而采用低精度的矩阵预处理方法实现加速是合理的.对于稀疏矩阵而言，为了不破坏矩阵的稀疏结构，预处理方法通常采用近似预处理形式，且由于矩阵具有对角占优性质，不需要对矩阵进行分析和预处理，使得预处理的时间被大幅缩短，细化迭代过程占据主要耗时.因此，本文的混合精度算法可以实现有效加速.表 3 展示了不同算例上 ILU 分解和矩阵求解过程的耗时，部分算例由于计时函数的精度限制未能展示.结果表明，ILU 分解耗时明显比双精度矩阵求解耗时要短，只占据了约 5%的时间耗时.



（a）混合精度实现加速效果　　（b）未实现加速效果

（a）Acceleration effects achieved　　（b）Without acceleration effect

图 5　算例求解时间

Fig. 5　Computation time costs for test cases

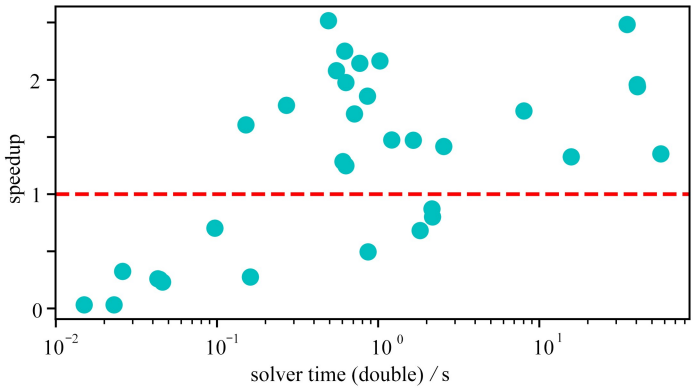


图 6　求解时间与加速比关系散点图

Fig. 6　The scatter plot of the computation time vs. the acceleration ratio

表 3　部分算例在预处理花费时间同总求解时间比较及精度比较

Table 3　Comparison of preprocessing time to total computation time and accuracy for selected test cases

name	ILU time /s	double precision time /s	mixed precision time /s	double precision RMSE	mixed precision RMSE
Poisson3Db	0.695	55.039	40.5	1.91E−11	4.59E−12
Venkat01	0.189	34.801	13.726	9.60E−14	3.54E−14
Raefsky1	0.092	2.526	1.757	6.91E−12	4.62E−12
Poisson3Da	0.09	7.855	4.426	1.85E−12	8.76E−13
Goodwin_013	0.013	1.133	0.772	3.20E−11	2.92E−13
shallow_water1	0.012	40.603	19.853	3.51E−13	6.14E−13
shallow_water2	0.012	40.087	19.937	2.30E−12	1.27E−13

图 7 中展示了矩阵维度大小同求解时间的关系图,其中横纵坐标均使用对数坐标轴.可以看出,求解时间基本随着矩阵维度增大而增大,部分算例由于矩阵条件数小,求解难度低,导致求解时间过短.我们认为这些算例上的加速空间及加速需求较小,因而在分析加速比影响时对这些样本进行了清理.

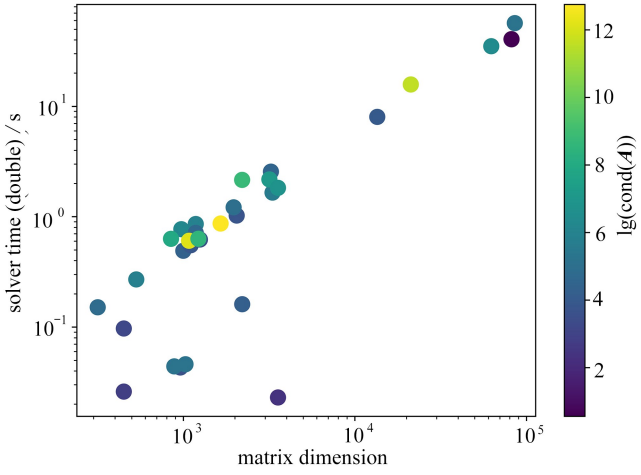


图 7 矩阵维度同求解时间关系图

Fig. 7 The relationship between the matrix dimension and the solution time

数据清理后,通过 Pearson 相关系数进行相关性分析,Pearson 相关系数常用来分析两变量之间的线性相关程度.本文构建了三种相关性系数模型:ρ_{xy},$\rho_{\lg(x),y}$,$\rho_{x,\lg(y)}$,其中 ρ_{xy} 表示原始变量与加速比的线性相关性;$\rho_{\lg(x),y}$ 表示了二变量间的对数相关性;$\rho_{x,\lg(y)}$ 表示两者间的指数相关性.结果如图 8 所示,其中前三个与迭代过程有关的变量均为双精度算法数据.

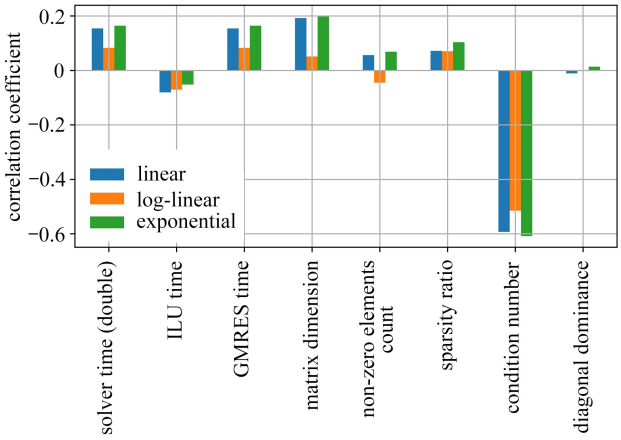


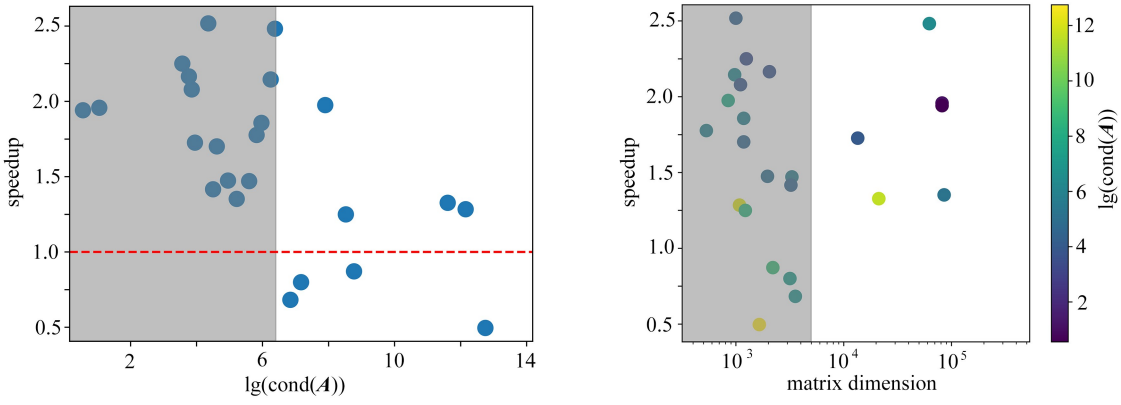
图 8 Pearson 相关系数分析结果

Fig. 8 Pearson correlation coefficient analysis results

从 Pearson 相关性结果可以看出,加速比的大小主要与矩阵条件数的相关性较强,且呈负相关性,这意味着当矩阵的条件数较大时,混合精度算法的加速性能减弱.我们分析这是由于低精度算法收敛性导致的.当矩阵条件数增大后,矩阵病态性增加.这意味着低精度的算法更加难收敛,需要更多的迭代次数,导致混合精度算法的加速性能下降.图 9(a)展示了矩阵条件数同加速比之间的关系,其中对条件数取了对数值,以更直观地看出两者的分布关系.结果表明,当条件数小于10^6 时,混合精度算法可以起到有效的加速效果;在条件数大于10^6 后,混合精度仍可能起到加速效果.这是因为条件数只表征了方程求解的病态性,求解的收敛速度还与方程的类型和初值条件等多种因素有关,但是其加速性能的鲁棒性下降.

图 9(b)是矩阵维度大小同加速比之间的散点图.其中颜色栏表示条件数的大小,灰色部分为矩阵维度小于 3 000 的样本数据.可以看到,在矩阵维度大于 3 000 后,所有样本点都实现了加速效果.当维度较小时,

条件数较小的样本基本也实现了加速效果.我们认为当矩阵维度增大后,求解时间增长.由于求解时间的主要耗时部分是 GMRES 迭代部分,即低精度迭代部分.因而混合精度算法能有更好的加速效果.



(a) 条件数　　(b) 矩阵维度

(a) The condition number　　(b) The matrix dimension

图 9　条件数、矩阵维度对加速比影响

Fig. 9　The influences of the condition number and the matrix dimension on the acceleration ratio

2.3　计算收敛性分析

进一步,本文探讨了条件数对混合精度算法的精度和收敛性的影响.图 10 展示了在混合精度求解器下,求解精度与条件数之间的关系.条件数是衡量矩阵病态程度的重要指标,它反映了矩阵在数值计算中的鲁棒性,对于线性方程组求解问题,矩阵条件数越大,线性方程组求解就越困难.结果表明,随着条件数的增大,混合精度和双精度的求解器的精度都会下降.在条件数小于10^9 时,混合精度算法基本可以达到双精度的精度要求.然而,当条件数较大时,混合精度算法可能出现未完全收敛情况,这在图 3 中的 ex20 算例中有所体现.在该情况下,双精度算法基本收敛到,而混合精度仅达到,未能满足精度要求.但在更高的条件数上,混合精度算法仍可能收敛到目标精度上.这说明本文算例中的条件数仍然在单精度浮点数的收敛范围内.但是,高条件数可能对混合精度算法的鲁棒性产生影响.

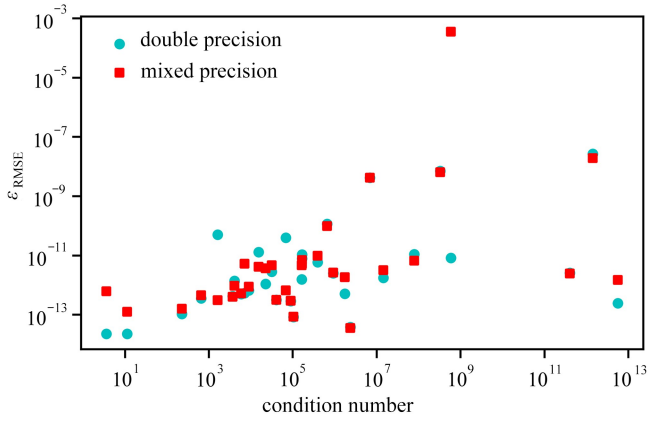


图 10　混合精度及双精度求解器的误差与条件数关系图

Fig. 10　The error vs. the condition number for mixed-precision and double-precision solvers

针对本文中出现的未能收敛的算例 ex20,我们进行更加详细的分析,以确定混合精度算法的收敛性能.表 4 中展示了不同迭代参数下的算例结果和加速比.结果表明,混合精度算法在 ex20 算例上可以实现收敛.但其收敛所需要的次数超出了先前所设计的最大迭代次数.因而导致混合精度计算的误差较大.我们分析认为这与原始矩阵的条件数较大且所需要的迭代次数本身较多有关.原始矩阵的条件数大小约为 6×10^8,属于

较大条件数范围,使用低精度算法需要更大的迭代次数才可能达到目标精度.且双精度算法迭代所需次数为433,收敛速度较慢.同时,当低精度迭代次数较少,每次细化迭代过程对原始解的校正量较少,也可能导致最终结果不能收敛到目标精度.由于 GMRES 内迭代时,单步迭代时间会随着迭代次数的升高而提升.采用较大的内迭代次数会导致计算成本的上升.本文建议细化迭代部分的最大迭代次数在双精度算法最大迭代次数基础上最多缩小 2~3 倍,同时需要保证有充足的外迭代次数,以保证结果可以收敛到精度需求.与前一小节的结论一致,由于矩阵的条件数超过了给定的加速指标,因而不同参数下的混合精度算法均未能起到有效加速效果.

表 4 未收敛算例不同迭代参数下求解精度和加速比结果

Table 4 Solutions precisions and speedup ratios of non-convergent numerical examples with different iteration parameters

case	solver precision	maximum outer iteration	maximum inner iteration	precision RMSE	iteration	CPU time/s
case 1	double	2(restart count)	300	8.18E-12	433	1.688
case 2	mixed	10	100	3.51E-4	1 000	2.203
case 3	mixed	10	200	8.49E-12	1 554	5.484
case 4	mixed	20	100	9.36E-12	1 600	3.859
case 5	mixed	40	50	9.289	2 000	3.248
case 6	mixed	4	500	2.415E-5	1 988	14.812 5

3 结 论

在流体仿真问题中,线性方程组的计算加速对提高 CFD 的计算效率有关键的作用.目前许多高性能的计算硬件使用了低精度的浮点数计算单元,大大提高了硬件性能,降低了芯片的功耗.但基于高精度浮点计算的数值方法无法充分利用这些高性能硬件.因此,混合精度算法受到了研究者的重视.基于稠密矩阵的混合精度算法已经有了较为广泛的研究,而系数矩阵的混合精度算法还缺乏有效的算法.本文基于现有研究,结合 CFD 问题背景,开展了针对 CFD 问题的混合精度算法的研究.

本文首先研究了问题背景中的待求解矩阵的数据特点,高度稀疏性和对角占优性质.其次,提出了基于稀疏存储架构下的混合精度算法,sparse GMRES-IR 算法.在单核 CPU 环境上,使用佛罗里达大学的公开稀疏矩阵数据集进行算例测试,发现所使用的混合精度加速算法可以实现最高 2.5 倍的加速效果,并在大部分算例上达到和双精度算法同量级的精度结果.针对未能实现加速的算例,研究了求解时间、问题的规模及矩阵的条件数对混合精度算法的加速比和精度的影响,发现所提出的混合精度算法主要在大规模矩阵上有良好的加速效果,在小规模矩阵上加速效果不显著.总体来说,本文所提出的算法有以下优势:

1) 基于稀疏存储的算法架构,本文算法采用了不完全的预处理方法和 CSR 稀疏存储格式.保证了整个求解过程中不使用任何稠密存储的矩阵,提高了计算效率,降低了内存消耗.

2) 针对 CFD 问题背景的算法设计.本文算法考虑了 CFD 时间推进矩阵的数值特点.并对预处理方法和求解器进行了特殊设计,提高了计算效率.

3)适用于低精度浮点计算单元.通过低精度浮点数计算程序中的主要耗时部分,发挥了低精度计算的计算效率优势,理论上在其他高性能计算硬件(如 TPU)上会有更大的加速潜能.

后续研究中:一方面,当前的研究仅使用了单双混合精度,下一步将加入半精度(FP16)浮点计算部分,并尝试使用不同的预处理方法,以此来扩大混合精度算法的适用范围.另一方面,当前的测试平台为单核 CPU,对于并行算法和 GPU、TPU 等其他方面的加速效果还有待研究.

参考文献(References):

[1] JIMÉNEZ J. Computing high-Reynolds-number turbulence: will simulations ever replace experiments? [J]. *Journal of Turbulence*, 2003, **4**. DOI:10.1088/1468-5248/4/1/022.

[2] CHOQUETTE J, GANDHI W, GIROUX O, et al. NVIDIA A100 tensor core GPU: performance and innovation [J]. *IEEE Micro*, 2021, **41**(2): 29-35.

[3] RAVIKUMAR A, SRIRAMAN H. A novel mixed precision distributed TPU GAN for accelerated learning curve [J]. *Computer Systems Science and Engineering*, 2023, **46**(1): 563-578.

[4] NOVITSKIY I M, KUTATELADZE A G. DU8ML: machine learning-augmented density functional theory nuclear magnetic resonance computations for high-throughput in silico solution structure validation and revision of complex alkaloids[J]. *Journal of Organic Chemistry*, 2022, **87**(7): 4818-4828.

[5] HAIDAR A, TOMOV S, DONGARRA J, et al. Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers[C]//*SC*18: *International Conference for High Performance Computing*, *Networking*, *Storage and Analysis*. Dallas, TX, USA: IEEE, 2018: 603-613.

[6] DU S, BHATTACHARYA C B, SEN S. Maximizing business returns to corporate social responsibility (CSR): the role of CSR communication[J]. *International Journal of Management Reviews*, 2010, **12**(1): 8-19.

[7] DENG L, LI G, HAN S, et al. Model compression and hardware acceleration for neural networks: a comprehensive survey[J]. *Proceedings of the IEEE*, 2020, **108**(4): 485-532.

[8] BAI Y, WANG Y X, LIBERTY E. ProxQuant: quantized neural networks*via* proximal operators[J/OL]. 2018 [2024-07-10]. https://arxiv.org/abs/1810.00861v3.

[9] BUTTARI A, DONGARRA J, KURZAK J, et al. Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy[J]. *ACM Transactions on Mathematical Software*, 2008, **34**(4): 1-22.

[10] 陈逸, 刘博生, 徐永祺, 等. 混合精度频域卷积神经网络 FPGA 加速器设计[J]. 计算机工程, 2023, **49**(12): 1-9.(CHEN Yi, LIU Bosheng, XU Yongqi, et al. FPGA accelerator design for hybrid precision frequency domain convolutional neural network[J]. *Computer Engineering*, 2023, **49**(12): 1-9.(in Chinese))

[11] AMESTOY P R, DUFF I S, L'EXCELLENT J Y. Multifrontal parallel distributed symmetric and unsymmetric solvers[J]. *Computer Methods in Applied Mechanics and Engineering*, 2000, **184**(2/3/4): 501-520.

[12] LI X S, DEMMEL J W. SuperLU_DIST: a scalable distributed-memory sparse direct solver for unsymmetric linear systems[J]. *ACM Transactions on Mathematical Software*, 2003, **29**(2): 110-140.

[13] HOGG J D, SCOTT J A. A fast and robust mixed-precision solver for the solution of sparse symmetric linear systems[J]. *ACM Transactions on Mathematical Software*, 2010, **37**(2): 1-24.

[14] CARSON E, HIGHAM N J. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems[J]. *SIAM Journal on Scientific Computing*, 2017, **39**(6): A2834-A2856.

[15] HIGHAM N J, PRANESH S. Exploiting lower precision arithmetic in solving symmetric positive definite linear systems and least squares problems[J]. *SIAM Journal on Scientific Computing*, 2021, **43**(1): A258-A277.

[16] LOE J A, GLUSA C A, YAMAZAKI I, et al. A study of mixed precision strategies for GMRES on GPUs[J/OL]. 2021[2024-07-10]. https://arxiv.org/abs/2109.01232v1.

[17] AMESTOY P, BUTTARI A, HIGHAM N J, et al. Five-precision GMRES-based iterative refinement[J]. *SIAM Journal on Matrix Analysis and Applications*, 2024, **45**(1): 529-552.

[18] HAIDAR A, BAYRAKTAR H, TOMOV S, et al. Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems[J]. *Proceedings of the Royal Society A*: *Mathematical*, *Physical and Engineering Sciences*, 2020, **476**(2243): 20200110.

[19] ZOUNON M, HIGHAM N J, LUCAS C, et al. Performance impact of precision reduction in sparse linear systems solvers[J]. *PeerJ Computer Science*, 2022, **8**: e778.

[20] GRATTON S, SIMON E, TITLEY-PELOQUIN D, et al. Exploiting variable precision in GMRES[EB/OL]. 2019 [2024-07-10]. https://arxiv.org/abs/1907.10550v2.

[21] GIRAUD L, HAIDAR A, WATSON L T. Mixed-precision preconditioners in parallel domain decomposition solvers[M]//*Lecture Notes in Computational Science and Engineering*. Berlin: Springer, 2008: 357-364.

[22] GÖBEL F, GRÜTZMACHER T, RIBIZEL T, et al. Mixed precision incomplete and factorized sparse approximate inverse preconditioning on GPUs[M]//*Lecture Notes in Computer Science*. Cham: Springer International Publishing, 2021: 550-564.

[23] 陈华, 史悦戎. 基于 GPU 的重启 PGMRES 并行算法研究[J]. 计算机工程与应用, 2014, **50**(7): 35-40.(CHEN Hua, SHI Yuerong. Study on restarted PGMRES parallel algorithm with GPU[J]. *Computer Engineering and Applications*, 2014, **50**(7): 35-40.(in Chinese))

[24] 冯选燕, 燕振国, 朱华君, 等. 非精确 Newton 方法中线性迭代收敛判据研究[J]. 空气动力学学报, 2023, **41**(12): 28-36.(FENG Xuanyan, YAN Zhenguo, ZHU Huajun, et al. Study on the convergence criterion of linear iteration in inexact Newton methods[J]. *Acta Aerodynamica Sinica*, 2023, **41**(12): 28-36.(in Chinese))

[25] 贡伊明, 刘战合, 刘溢浪, 等. 时间谱方法中的高效 GMRES 算法[J]. 航空学报, 2017, **38**(7): 120894.(GONG Yiming, LIU Zhanhe, LIU Yilang, et al. Efficient GMRES algorithm in time spectral method[J]. *Acta Aeronautica et Astronautica Sinica*, 2017, **38**(7): 120894.(in Chinese))

[26] 伍康, 吕毅斌, 石允龙, 等. 有界多连通区域数值保角变换的 GMRES(m)法[J]. 应用数学和力学, 2022, **43**(9): 1026-1033.(WU Kang, LÜ Yibin, SHI Yunlong, et al. The GMRES(m) method for numerical conformal mapping of bounded multi-connected domains[J]. *Applied Mathematics and Mechanics*, 2022, **43**(9): 1026-1033.(in Chinese))

[27] 肖文可, 陈星玎. 求解 PageRank 问题的重启 GMRES 修正的多分裂迭代法[J]. 应用数学和力学, 2022, **43**(3): 330-340.(XIAO Wenke, CHEN Xingding. A modified multi-splitting iterative method with the restarted GMRES to solve the PageRank problem[J]. *Applied Mathematics and Mechanics*, 2022, **43**(3): 330-340.(in Chinese))